Intel® XML Software Suite 1.2 for C/C++

User's Guide

Copyright © 2007–2008 Intel Corporation

All Rights Reserved

Document Number: 317576-005US

Revision: 1.5

World Wide Web: http://www.intel.com

Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright© 2007-2008, Intel Corporation. All rights reserved.

Revision History

Document Number	Revision Number	Description	Revision Date
317576	002	Intel® XML Software Suite 1.0 for C/C++	December 2007
317576	004	Intel® XML Software Suite 1.1 for C/C++	August 2008
317576	005	Intel® XML Software Suite 1.2 for C/C++	October 2008

Table Of Contents

1 Overview 1
About This Document 1
Product Overview 1
Supported Environments 3
Major Features 3
Thread Safety 5
Technical Support 7

2 Getting Started 9
Quick Start 9
Getting Version Information 10
Configuring the Intel® XML Software Suite 11
Working with the Intel® XML Software Suite 12
Sample Applications 23

3 Using the Intel® XML Parsing Accelerator 25
Parsing Data in SAX Mode 25
Parsing Data in DOM Mode 27
Parsing Data in XMLDocument Mode 28
Parsing Data with Schema Validation 30
Configuring the Intel® XML Parsing Accelerator 31

4 Using the Intel® XML Schema Accelerator 32
Validating Data 32
Configuring the Intel® XML Schema Accelerator 36

5 Using the Intel® XSLT Accelerator 38
Performing the XSL Transformation 38
Customizing the Intel® XSLT Accelerator 40
Object Types 41
XSLT Extensions 42

6 Using the Intel® XPath Accelerator 49
Performing the XPath Evaluation 49
Configuring the Intel® XPath Accelerator 52

7 Troubleshooting 56

Appendix A: Acronyms and Definitions 58

Appendix B: Compatibility Modes: Support for various Interpretations of XSL Conditions 59

Appendix C: References 66

Index 67

1 Overview

About This Document

This document describes the Intel® XML Software Suite, with an overview of its major features and a high-level view of the API and intended usage. With this document, you will be able to:

- start using the Intel XML Software Suite
- run the samples supplied with the package
- employ advanced functionality of the libraries making up the suite
- troubleshoot product-related issues.

The document is targeted at developers that need to perform XML data processing in their applications. The document assumes knowledge of XML and C/C++ programming skills.

Note

When viewing this guide in PDF format, your Adobe* Reader* viewer might generate hyperlinks from URL-formatted text of the guide. To disable this option, modify your preferences to disable automatic hyperlink generation.

Product Overview

The Intel® XML Software Suite for C/C++ is an XML processing library that you can use to perform parsing, validation, XPath search, and XSLT transformation of the XML data used in your application. The library implements a custom-designed API that closely mirrors the JAXP [15] API used for XML processing in Java* environments for functions where no standard C/C++ processing API is available. The API allows you to realize the best performing solution for your application without changing it.

This product provides rich, industrial-strength XML processing with exceptional performance. Now you can obtain the best that the Intel® architecture can offer and take advantage of such new features as the character and string processing instructions in the Intel® Streaming SIMD Extensions 4.2 (Intel® SSE 4.2), and scalability across multiple cores.

The Intel XML Software Suite covers all major stages of XML data processing, with each stage handled by separate components, as follows.

- **The Intel® XML Parsing Accelerator** parses the data to a Simple API for XML (SAX) API interface, a W3C Document Object Model (DOM) or a proprietary XMLDocument model:

    - Simple API for XML (SAX) model represents data as a sequence of events.

    - Document Object Model (DOM) model represents data as a node tree

    - `XMLDocument` represents data as an internal data stream for effective XML data processing in other components of the Intel XML Software Suite.

      After the document is parsed, the data can be further processed by other components in the library.

- **The Intel® XML Schema Accelerator** validates an XML document against a specified W3C XML [7] *schema* document. XML Schemas define a set of validation constraints which can be used to ensure that the XML data has the correct structure and values for your applications use.

- **The Intel® XSLT Accelerator** transforms an XML document into another XML, XHTML or text document by applying a W3C XSLT 1.0 stylesheet. The stylesheet describes how to transform the input document to the output document format.

- **The Intel® XPath Accelerator** navigates XML data by evaluating an XML path (W3C XPath [8]) expression over an XML document or a DOM tree. The expressions return references to parts of the document or values extracted from the document for your application to use.

The Intel® XML Software Suite can be plugged into different environments to accelerate XML processing, for example:

- Enterprise service busses. They introduce intermediaries between the client and the back-end business service that may parse the XML data passed in, validate that it conforms to a schema, perform an XSLT transformation to change and augment its format, and extract routing information using XPath.

- Application servers. They may provide web services stacks, perform data mashups, and much more. In web services stacks, messages may be parsed, validated and transformed

The Intel® XML Software Suite offers various interfaces to the underlying XML data representation, providing functionality for SAX, StAX or DOM parsing, schema or DTD validation, XSLT transformation, and XPath expression evaluation. It is recommended to use the suite for all the functions required, instead of combining functional elements with other implementations. This can result in significantly greater efficiency gains.

Supported Environments

The Intel® XML Software Suite for C/C++ is supported on Windows* and Linux* operating systems on IA-32 and Intel® 64 architectures. The following list includes the operating systems supported in this release:

- The Linux* OS:
    - Red Hat* Enterprise Linux* Server and Advanced Platform 5.0
    - Red Hat* Enterprise Linux* AS 4.0 - 2.6 kernel
    - Red Hat* Enterprise Linux* ES 4.0 - 2.6 kernel
    - Red Hat* Enterprise Linux* AS 3.0 - 2.4 kernel
    - Red Hat* Enterprise Linux* ES 3.0 - 2.4 kernel
    - SUSE* Linux* Enterprise Server 10 - 2.6 kernel
    - SUSE* Linux* Enterprise Server 9 - 2.6 kernel
- The Windows* OS:
    - Windows Server 2003 Standard*
    - Windows Server 2003 DataCenter*
    - Windows Server 2003 Enterprise*
    - Windows Server 2008*
    - Windows XP*
    - Windows Vista*

NOTE

The Intel XML Software Suite achieves the best performance on the Intel® architectures, by using features of the Intel® platforms and compiler tuning. Additionally, new software algorithms enable large performance gains on other platforms as well. If you need support for the platforms not listed above, send a request to xmlinfo@intel.com.

Major Features

All libraries of the Intel® XML Software Suite possess the following key features:

- *Performance*: The libraries of the Intel XML Software Suite demonstrate high performance. See the Intel® XML Software Products web site [13] for details.

- *Portable and Platform-Specific Performance:* the libraries making up the suite can use the available features of the underlying hardware platform. On the Intel® Core TM i7 with Intel® Streaming SIMD Extensions 4.2 (Intel® SSE 4.2) that accelerate string and character processing, or Intel® platforms supporting multiple cores, you can get significant performance gains without changing the software configuration. The Intel SSE 4.2 use two forms of parallelism:

    - performing operations on multiple 8- or 16-byte characters at a time, using the Intel Architecture's 256-bit XMM registers.

    - performing several operations concurrently on the array of characters.

- *Conformance and Compatibility*: The libraries of the Intel XML Software Suite conform to the following standards defining XML data processing:

    - W3C* XML 1.0 standard [1]

    - W3C Namespaces in XML 1.0 standard [2] except "Qualified Names in Declarations"

    - W3C DOM level 2.0 core standard [4] and partially DOM level 3 core with the limitations and specifics defined in the product *Release Notes*

    - SAX 2.0 specification [5] Core as defined in the product *Release Notes*

In addition to these, the components making up the suite possess a number of specific features, as described in Table 1.

Table 1. Component-Specific Features

Component	Conformance	Key characteristics	Supported file size
Intel® XML Parsing Accelerator		• Operates with data in the format of data streams, DOM trees, SAX events, or internal data streams • Supports encodings UTF-8, UTF-16, UTF-32, ISO-8859-1 and ASCII • Can enable validation of documents with the	On a 32-bit system, up to 20GB in SAX mode or 300MB in DOM mode; on a 64-bit system, up to 20GB in SAX mode or 600MB in DOM mode

Component	Conformance	Key characteristics	Supported file size
		Intel® XML Schema Accelerator before passing data to the application • Provides significant performance gains on the Intel® Core™ i7 with the Intel SSE 4.2.	
Intel® XML Schema Accelerator	W3C* XML Schema 1.0 standard [7] except PSVI (post-schema-validation infoset)		Up to and above 1GB
Intel® XSLT Accelerator	W3C XSLT 1.0 standard [3]; namespace declaration formats specific for the Apache* Xalan* XSLTC implementation [10] processor; OASIS* conformance test suite [6] for the Apache* Xalan-Java* processor	• Operates with data in the format of data streams, DOM trees, SAX events, or `XMLDocument` • Supports XSLT extensions, including EXSLT extension functions [12] to allow applications to extend XSLT functionality • Supports parallel transformation of an XML document with multiple threads • Provides significant performance gains on the Intel® Core™ i7 with the Intel SSE 4.2.	Up to and above 1GB
Intel® XPath Accelerator	W3C XML Path Language (XPath) 1.0 [8]		Up to and above 1GB

Thread Safety

In the current version of the Intel® XML Software Suite, certain structures and methods are thread-safe, while others are not. To ensure thread safety, application

code must follow specific rules when using the library interfaces. For example, multiple threads cannot recursively call `evaluate()` for one `XPath` or `XPathExpression` object because they are not thread-safe or re-entrant.

The general rule is that you can create and release instances of required objects from a shared factory in multiple threads. However, you can use the same instance of an object or configure a factory instance with setter methods in only one thread. For classes and methods not mentioned in this section, assume that they are not thread-safe.

Table 2 lists specific structures and methods grouped by their thread safety or unsafety. The table refers to factory classes of the C++ API, including factory classes that are responsible for specific objects, as follows:

- `DOMImplementationFactory` creates `DOMImplementation`.
- `SAXParserFactory` creates `SAXParser`.
- `SchemaFactory` creates `Schema`.
- `TransformFactory` creates `Templates`.
- `XMLDocFactory` creates `XMLDocParser` and `XMLDocWriter`.
- `XPathFactory` creates `XPath`.

Table 2. Distribution of Thread Safe and Unsafe Classes and Methods

Entity	Thread-safe	Thread-unsafe
Classes	• `Schema` • `Templates` • `DOMImplementation`	• `DOMParser` • `DOMWriter` • `SAXParser` • `XMLDocParser` • `XMLDocWriter` • `Transformer` • `Validator` • `XPath` • `XPathExpression`
Instantiating methods of factory classes	• `newInstance()` • `releaseInstance()`	`set*()`
Creator methods of factory classes	• `getDOMImplementation()` • `createSAXParser()`	

Entity	Thread-safe	Thread-unsafe
	• `createXMLDocParser()` • `createXMLDocWriter()` • `createSchema*()` • `newXPath()` • `newTemplates()`	
Releasing methods of factory classes	• `releaseSAXParser()` • `releaseXMLDocParser()` • `releaseXMLDocWriter()` • `releaseSchema()` • `releaseXPath()` • `releaseTemplates()`	

For detailed class and function definitions, see the API Reference shipped with the product.

Technical Support

To receive technical support for the Intel® XML Software Suite, register for an Intel Premier Support account at the Registration Center.

To register for an account, visit the Intel® Registration Center web site. If you have forgotten your password, send a request to: quadsupport@mailbox.intel.com. Please do not send your technical issue to this e-mail address.

Additional information and support are available from the Intel® Software Network-XML User Forum and XML Software Blog.

To submit an issue via the Intel® Premier Support web site, do the following:

1. Make sure that Java* and JavaScript* are enabled in your browser.
2. Go to http://premier.intel.com and log in.
3. Submit your issue:
   a. Click **Submit Issues** in the left channel.
   b. Select the product type from the **Product Type** drop-down menu. For the Intel® Software Products, choose **Development Environment** (tools, SDV, EAP) as the product type.
   c. Select the product for which you want to submit the issue from the **Product Name** drop-down menu.

d. Enter a headline-type description of the problem or question in the **Issue Title** text box.

e. Enter a detailed question, or a detailed problem description in the **Question** text box.

f. Select **Request Type** from the drop-down menu.

g. To complete the submission process, click **Submit Issue** or **Submit Issue/Upload Files**, if you need to attach a file, for example, a test case or a screenshot. You will get an issue number that enables you to track your issue.

2 Getting Started

Quick Start

This section of the user guide explains how to quickly set up the library for use in your environment.

After installing the Intel® XML Software Suite, as described in the *Installation Guide*, do the following:

1. Check that the content of the installation directory matches the description in the `README` file.

   If discrepancies occur, consult the Troubleshooting section for recommendations on resolving the issue.

2. On the Linux* OS, configure your environment settings by running configuration scripts from the `<installation_dir>/tools` directory.

3. Check that your environment is set up correctly.

   The environment variables should be set as shown in Table 3.

Table 3. Environment Settings

OS/ Architecture	Variable Name	Linux*	Windows*
IA-32	INCLUDE	**install_dir** /include	**install_dir** \Include
	PATH	N/A	**install_dir** \Bin\ia32
	LD_LIBRARY_PATH	**install_dir** /bin/ia32	N/A
	LIB	N/A	**install_dir** \Lib\ia32
Intel® 64	INCLUDE	**install_dir** /include	**install_dir** \Include

OS/ Architecture	Variable Name	Linux*	Windows*
	PATH	N/A	**install_dir** \Bin\intel64
	LD_LIBRARY_PATH	**install_dir** /bin/intel64	N/A
	LIB	N/A	**install_dir** \Lib\intel64

After setting up the environment, you are ready to start using the Intel XML Software Suite. You can also adjust the product behavior by changing configuration settings, see Configuring the Intel® XML Software Suite.

Getting Version Information

The Intel® XML Software Suite enables you to get the library version information. There are two methods defined in `<installation_dir>/include/common/version.h` to get this information:

1. `getXSSVersion()` to obtain the XSSVersion structure which contains the version information
2. `getXSSVersionString()` to extract a whole version description string

The version structure is defined as follows:

```
 typedef struct {

    int          MajorVersion;    ///< The major number of the
current library version

    int          MinorVersion;    ///< The minor number of the
current library version

    int          PatchVersion;    ///< The patch number of the
current library version

    int          BuildNumber;     ///< The internal build number

    XMLChar *    Platform;        ///< The platform which the
current library fit for

} XSSVersion;
```

See also:

Version Query Samples

Configuring the Intel® XML Software Suite

The file `config.xml` , which installs to `<installation_dir>/conf`, enables you to control behavior of the Intel® XML Software Suite. For example, you may want to increase or decrease the memory threshold to speed up data processing or to reduce memory consumption accordingly.

After successfully installing the Intel XML Software Suite, you can delete, re-create, or modify the configuration file as needed. With the current version of the configuration file, you can:

- Adjust the *threshold of memory usage* to restrict the amount of system virtual memory consumed by the Intel XML Software Suite. The default value is 512MB.

- Configure `DocumentBuilderFactory` to ignore all unsupported features and attributes. By default, `DocumentBuilderFactory` throws exceptions for unsupported features and attributes. To ignore the unsupported features and attributes, set the throw-unknown-feature value as false.

```
<component name ="Intel XSLT Accelerator">
<property name="namespace-prefix" value="false" />
</component>
```

- Configure *parallel XSLT transformation* to allow the Intel® XSLT Accelerator to manipulate an XML document in multiple threads. The value of this option must be an integer. The default value is 0, so the feature is disabled; a larger value enables parallel transformation and sets the number of maximum working threads. However, the Intel XSLT Accelerator can further adjust the maximum number of threads; for instance, if you set this option to 128, the Intel XSLT Accelerator may choose to create only 4 threads for optimal performance.

The settings in the configuration file can apply to all or specific components, and have the following format:

```
<property name="name_of_property" value="current_value" />
```

Component-specific settings are inside respective component tags, for example:

```
<component name ="Intel XSLT Accelerator">
<property name="maximum memory" value="512m" />
```

```
</component>
```

On startup, the Intel XML Software Suite loads and parses `config.xml` to get the settings. If no correct value is found or if an error occurs, the product uses the default built-in value.

See also:

- Configuring the Intel® XML Parsing Accelerator
- Configuring the Intel® XML Schema Accelerator
- Configuring the Intel® XSLT Accelerator
- Configuring the Intel® XPath Accelerator

Working with the Intel® XML Software Suite

Working In Eclipse* IDE

The Intel® XML Software Suite for C/C++ can be used in Eclipse* IDE with C++ Development Tooling (CDT) plug-ins on Linux* and Windows* IA-32 and Intel® 64 architectures.

Prerequisites:

To start using the Intel XML Software Suite in Eclipse IDE, install the following components:

For the Windows* OS:

- Eclipse IDE 3.1.1
- CDT 3.0.2 plug-in

For the Linux* OS:

- Eclipse IDE 3.3.2
- CDT 4.0 plug-in

See Also

Working in Eclipse* IDE on the Linux* OS

Working in Eclipse* IDE on the Windows* OS

Working in Eclipse* IDE on the Linux* OS

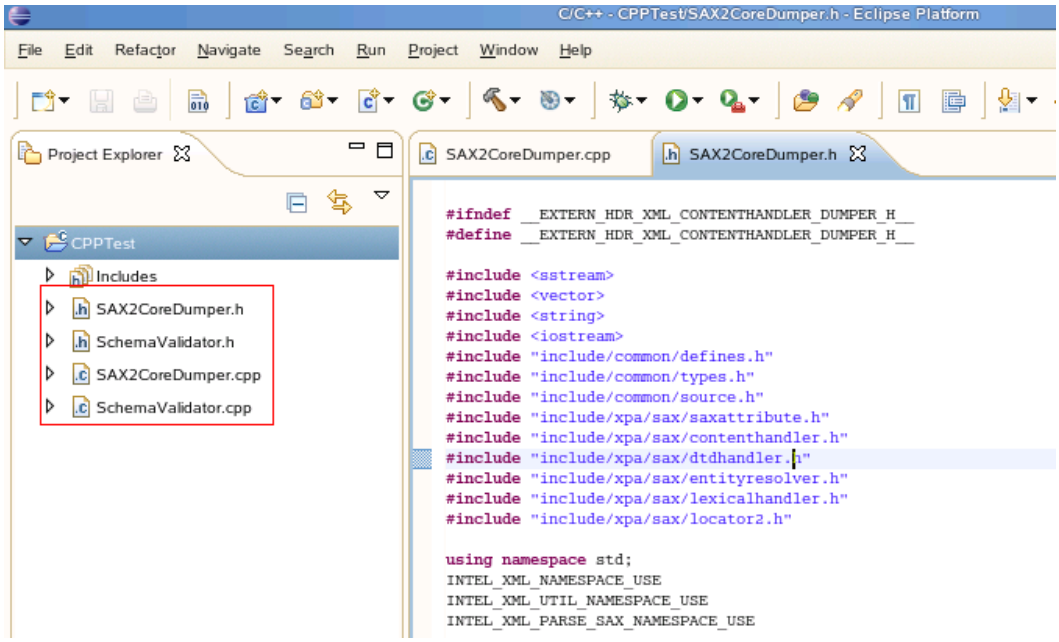
This section gives step-by-step instructions on how to use the Intel® XML Software Suite with Eclipse IDE on Linux IA-32 architecture. The steps are analogous for the Intel® 64 architecture. The installation package of the Eclipse IDE and CDT used in this example is *eclipse-cpp-europa-winter-linux-gtk.tar.gz*. You can download it from http://www.eclipse.org/downloads/. The source files used are copied from the schema validation sample in `examples/xsa/SchemaValidator` directory, and are as follows:

- `SchemaValidator.h`
- `SchemaValidator.cpp`
- `SAX2CoreDump.h`
- `SAX2CoreDump.cpp`

In the screenshots, all relevant settings are marked with red rectangles.

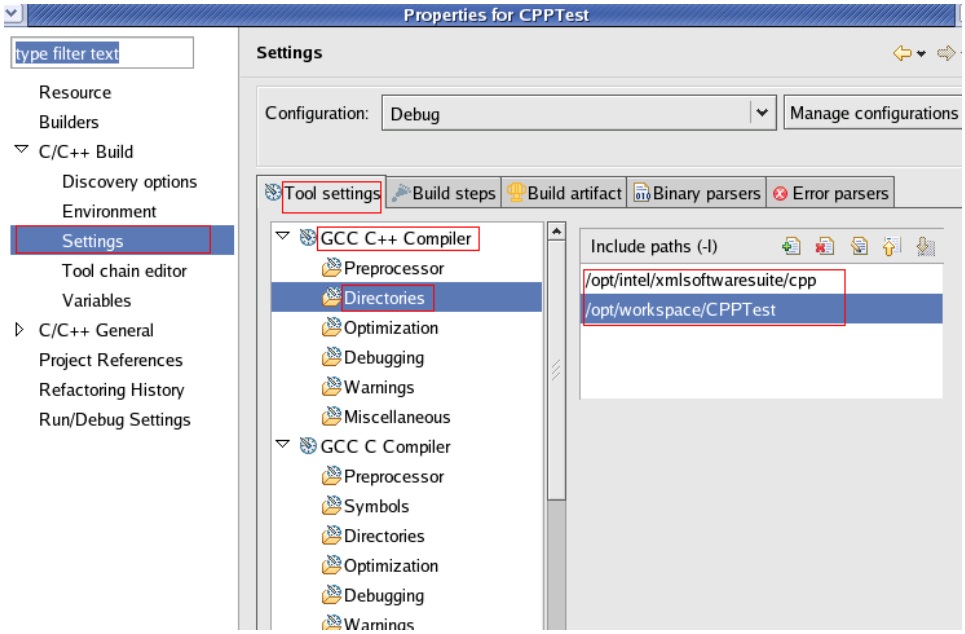
Follow these steps to start using the Intel XML Software Suite in Eclipse IDE with CDT:

1. Create a C++ project and set the project type to executable. This is a common step for all C++ projects.
2. Add source files, including CPP source files and header files.

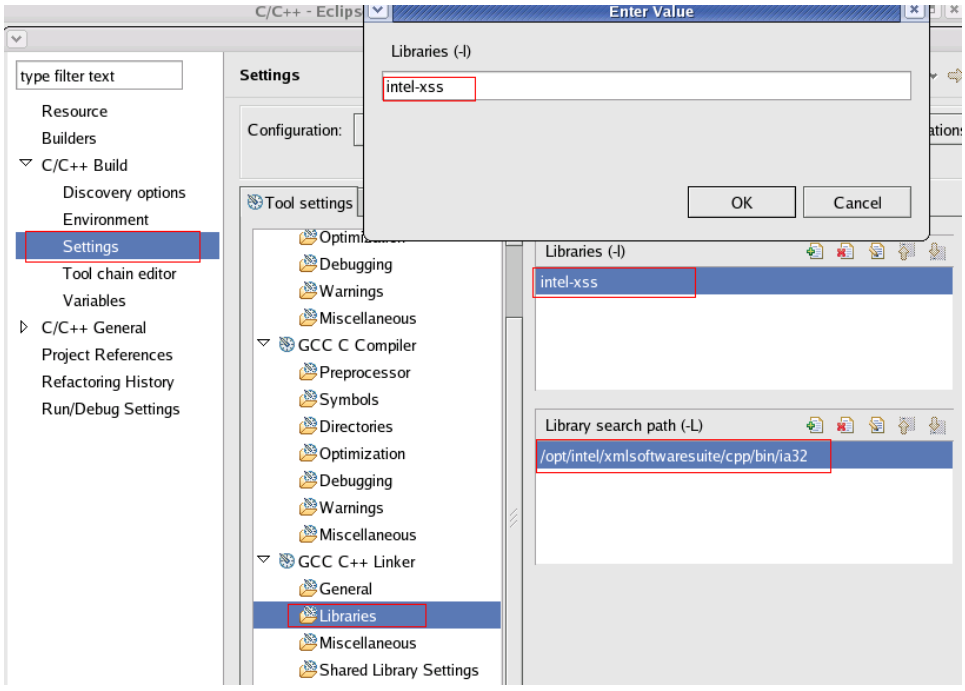


3. Set the *include* path to the CPP package install directory and the path to the current project.

For that, go to **C/C++ Build** > **Settings** > **Tool Settings** > **Directories** > **Include paths** and click the "**add**" icon to add the two paths one by one.



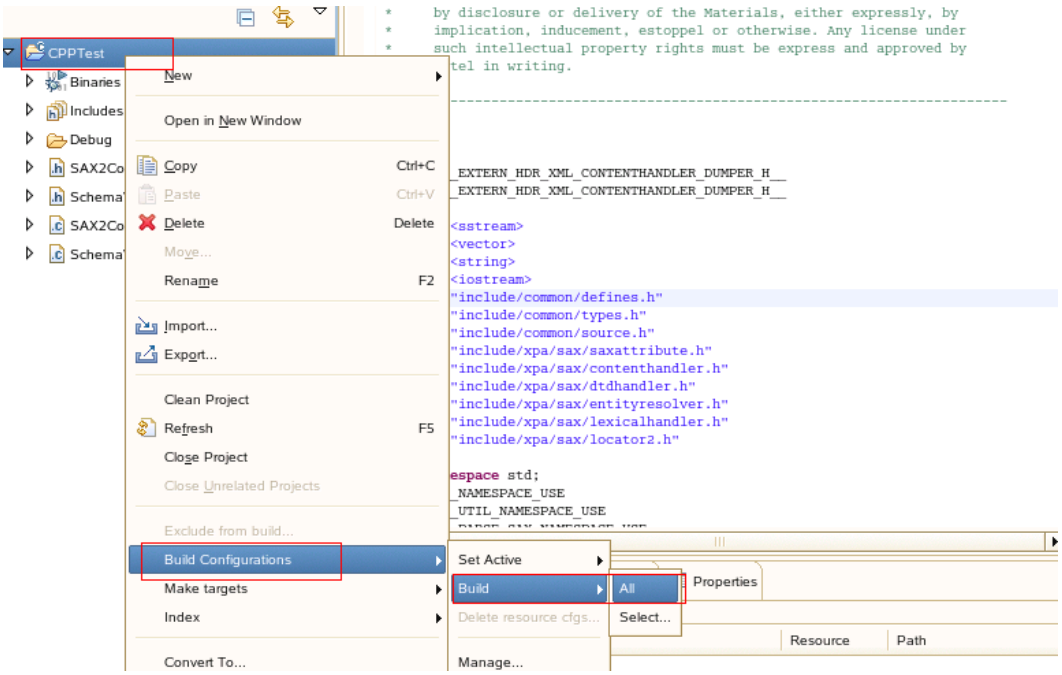
4. Set the library path. For that, go to **C/C++ Build** > **Settings** > **GCC C++ Linker** > **Libraries**. In the **Library search path** dialog, click the "**add**" icon to add a new path.



NOTE

The library name is `intel-xss`; the `lib` prefix and the `.so` postfix are generated automatically.

5. Right-click the project and select **Build configurations > Build > All** to build the project.



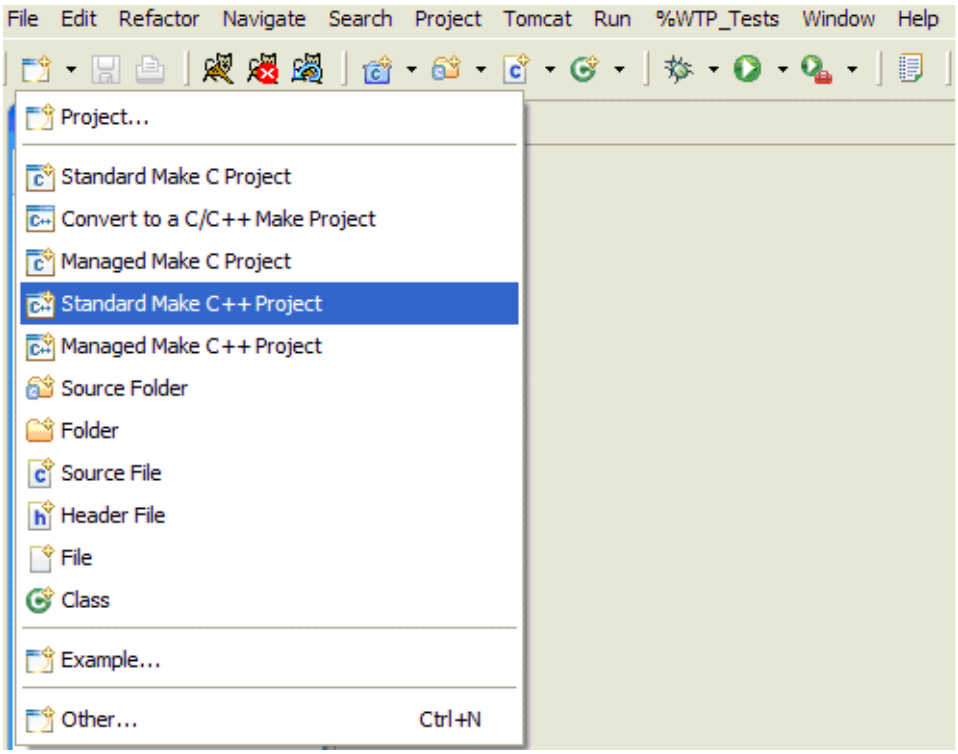
Working in Eclipse* IDE on the Windows* OS

This section illustrates how to use the Intel XML Software Suite in Eclipse* IDE on the Windows XP* operating system. The steps on other Windows platforms are analogous. The installation package of the Eclipse IDE and CDT used in this example is Eclipse 3.1.1 and CDT 3.0.2. In fact, any combination of Eclipse IDE and CDT can work. Here "nmake" is used, and this tool is installed by default when Microsoft* Visual Studio 2003* is installed.

In the screenshots, all relevant settings are marked with the red rectangles.

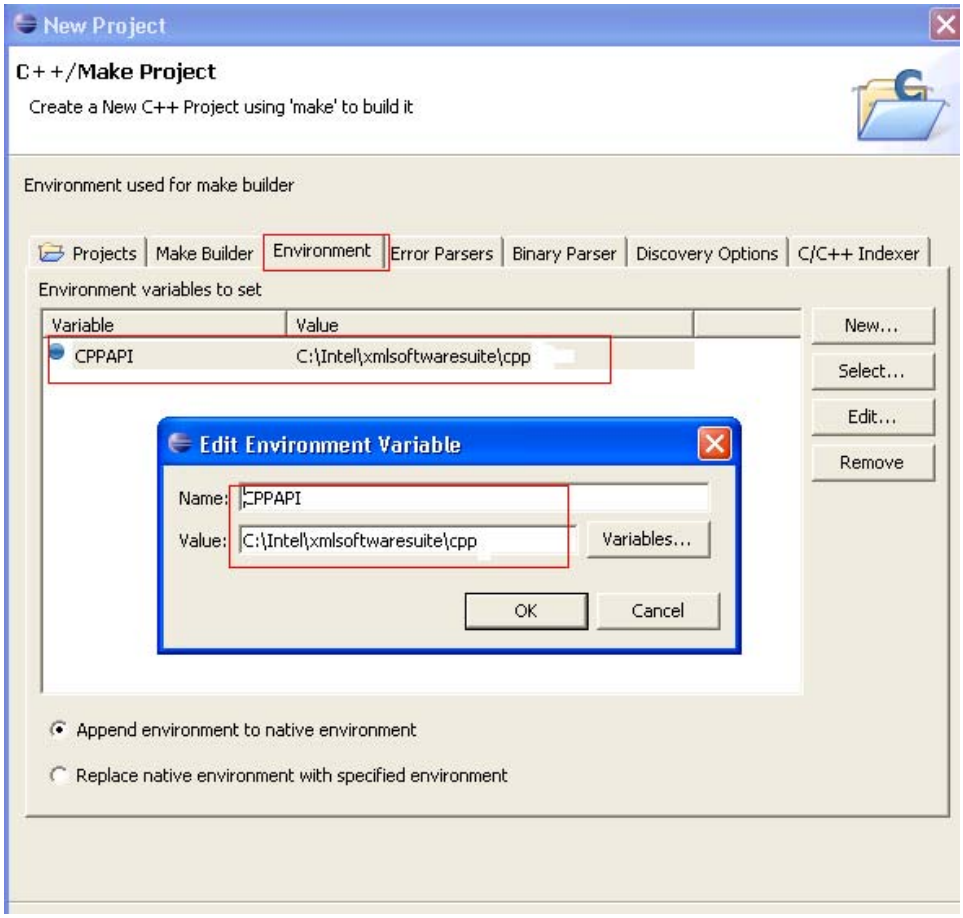
To start using the Intel XML Software Suite in Eclipse IDE with CDT, follow these steps:

1. Go to **Project** > and select **Standard Make C++ Project**, as shown:

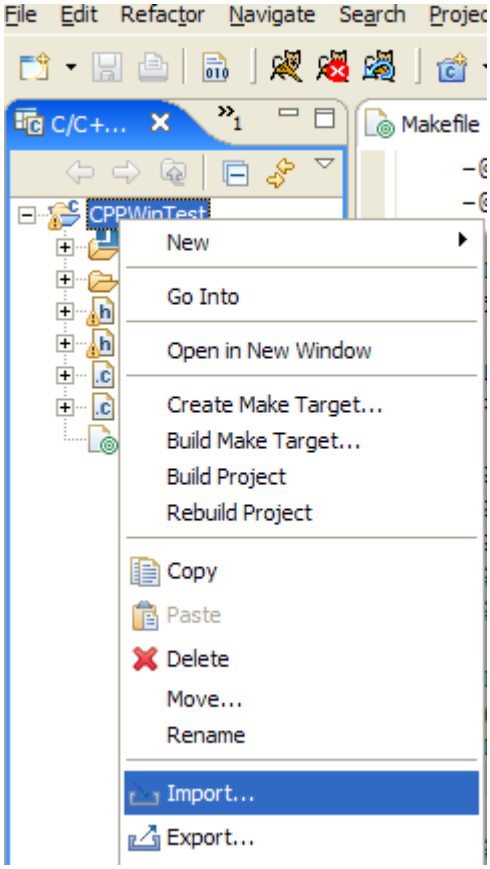


2. Set the project name and location.

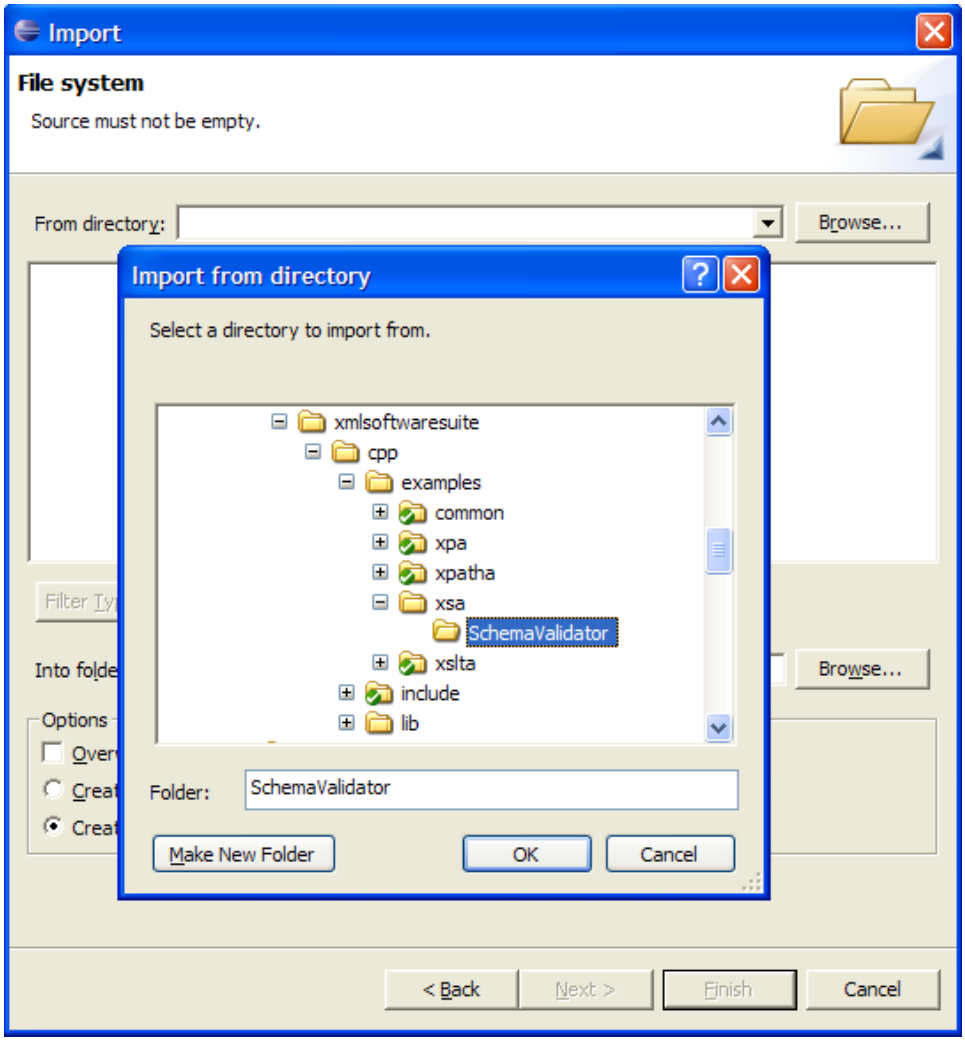
3. Set the project properties. Under **Build command**, enter `nmake`. Set an environment variable as `CPPAPI`.

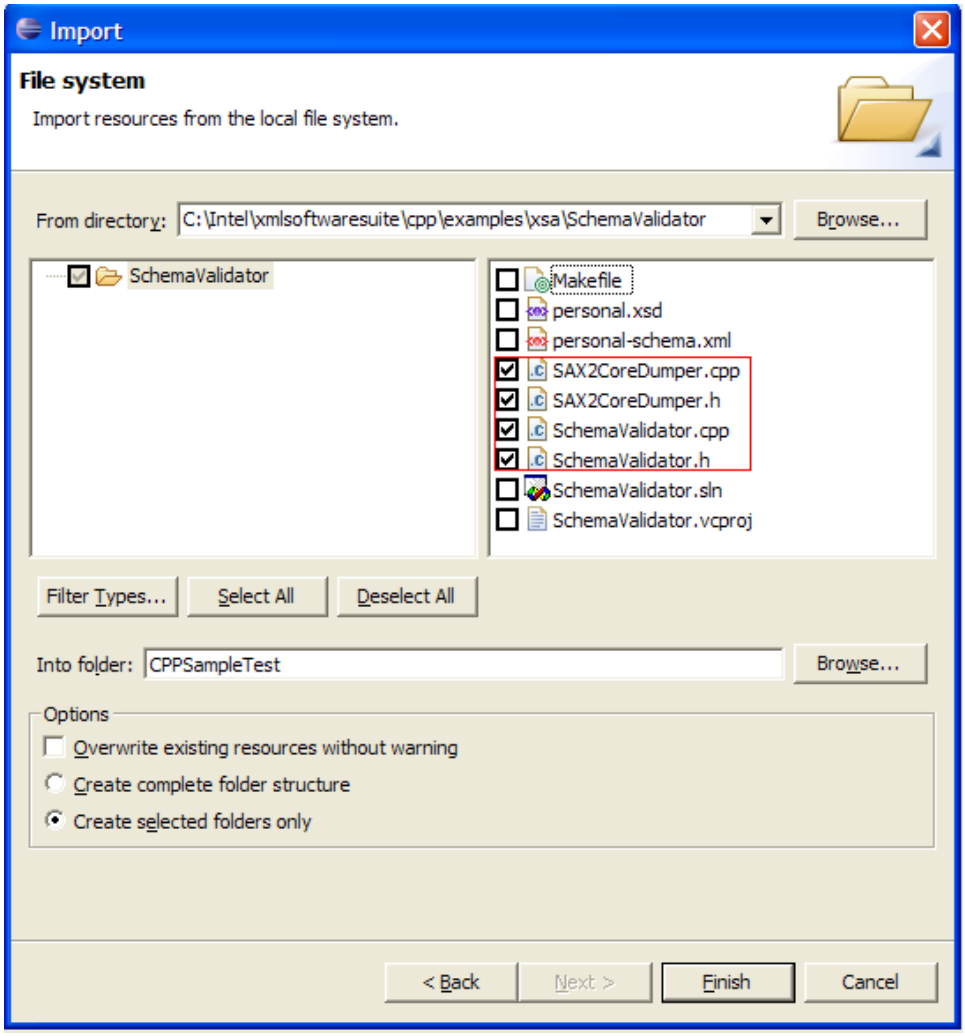


4. Import the source file to the project.
   Right click the project and choose **Import...**.



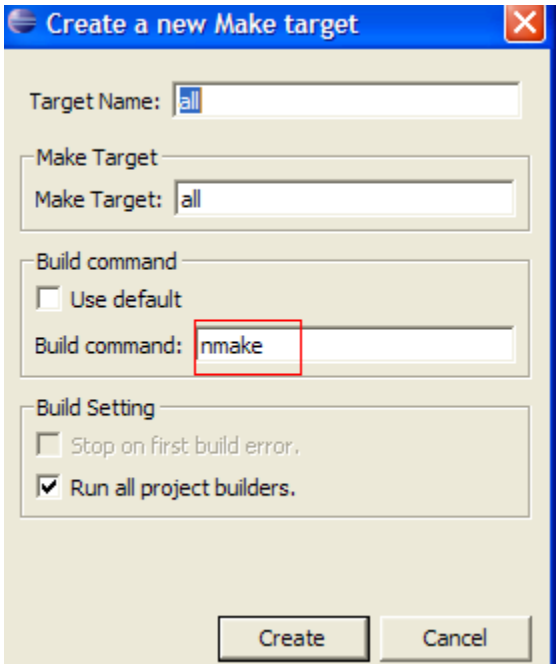
5. Choose the example directory **SchemaValidator** from **cpp** > **examples** > **xsa**, and then choose the files to import:


Import
File system
Source must not be empty.
From directory:
Browse...
Import from directory
Select a directory to import from.
xmlsoftwaresuite
cpp
examples
common
xpa
xpatha
xsa
SchemaValidator
xslta
include
lib
Folder:
SchemaValidator
Make New Folder
OK
Cancel
Into folde
Browse...
Options
< Back
Next >
Finish
Cancel



6. Create a Makefile for the project, and add the following contents to the Makefile.

Refer to MSDN* for details of the format in this Makefile.




```
          OUTDIR=.\Debug
          INTDIR=.\Debug
          # Begin Custom Macros
          OutDir=.\Debug
          # End Custom Macros
          ALL : "$(OUTDIR)\SchemaValidator.exe"
          CLEAN :
          -@erase "$(INTDIR)\SAX2CoreDumper.obj"
          -@erase "$(INTDIR)\SchemaValidator.obj"
          -@erase "$(INTDIR)\SchemaValidator.exe"
          -@erase "$(INTDIR)\SchemaValidator.ilk"
          -@erase "$(INTDIR)\SchemaValidator.pdb"
          "$(OUTDIR)" :
          if not exist "$(OUTDIR)/$(NULL)" mkdir
"$(OUTDIR)"
          CPP=cl.exe
          CPP_PROJ=/nologo /MTd /W3 /Gm /GX /ZI /Od /I
$(CPPAPI) /D "WIN32"
/D "_DEBUG" /D "_CONSOLE" /D "_MBCS" /Fo"$(INTDIR)\\"
/Fd"$(INTDIR)\\" /FD /GZ /c
          LINK32=link.exe
```

```
          LINK32_FLAGS=intel-xss.lib kernel32.lib
user32.lib gdi32.lib winspool.lib
comdlg32.lib advapi32.lib shell32.lib ole32.lib
oleaut32.lib uuid.lib
odbc32.lib odbccp32.lib /nologo
/subsystem:console/incremental:yes/pdb:"$(OUTDIR)\SchemaVal
idator.pdb"
/debug/machine:I386/out:"$(OUTDIR)\SchemaValidator.exe"/lib
path:$(CPPAPI)\lib\ia32
          LINK32_OBJS= \
          "$(INTDIR)\SchemaValidator.obj" \
          "$(INTDIR)\SAX2CoreDumper.obj"
          "$(OUTDIR)\SchemaValidator.exe" : "$(OUTDIR)"
$(DEF_FILE)
 $(LINK32_OBJS)
           $(LINK32) @<<
           $(LINK32_FLAGS) $(LINK32_OBJS)
           <<
           .cpp{$(INTDIR)}.obj::
           $(CPP) @<<
           $(CPP_PROJ) $<
           <<
           SOURCE=SAX2CoreDumper.cpp
           "$(INTDIR)\SAX2CoreDumper.obj" : $(SOURCE)
"$(INTDIR)"
           $(CPP) $(CPP_PROJ) $(SOURCE)
           SOURCE=SchemaValidator.cpp
           "$(INTDIR)\SchemaValidator.obj" : $(SOURCE)
"$(INTDIR)"
           $(CPP) $(CPP_PROJ) $(SOURCE)
```

7. Create the Make target.
   Right click the Makefile and choose **Create Make Target…**. Change the build command to **nmake**.

8. Run the Makefile.
   Right click the Makefile and choose **Build Make Target…**, and choose the target you have set.

Sample Applications

The current distribution of the Intel® XML Software Suite includes multiple source code examples that show how you can use the product with your application. The examples are packaged with source files, `makefiles`, and sample input. You can run the examples with the provided sample files or with your own files.

This topic lists the examples, shipped with the current version of the product. For detailed instructions on building and running each sample application, please see the *Samples Guide*.

Samples are divided into several groups by the component they belong to.

For the Intel® XML Parsing Accelerator

DOMPrint	Parses an XML document in DOM mode and serializes the DOM tree to standard output.
DOMTraverse	Parses an XML document in DOM mode and traverses all nodes in the created DOM tree.
SAX2Print	Parses an XML document in SAX mode and dumps the content to standard output.
XMLDocPrint	Parses an XML document in `XMLDocument` mode and dumps the content to standard output.
ValidatingParser	Parses an XML document with schema validation.

For the Intel® XML Schema Accelerator

SchemaValidator	Validates XML documents in the Stream, SAX, DOM or `XMLDocument` mode against the Schema XML documents.

For the Intel® XSLT Accelerator

SimpleTransform	Performs stream-to-stream transformation for the specified `xml` and `xsl` files.
MultipleTypeTransform	Performs transformation with various input and output formats.
TransformChain	Performs pipeline transformation using the Intel proprietary format `XMLDocument` as intermediate input and output.
StylesheetParameter	Sets and gets stylesheet parameters, see Configuring the

	Intel® XSLT Accelerator.
CompileStylesheet	Compiles the stylesheet supplied in various source formats.
UDF	Employs user-defined extension functions in C++ during XSLT transformation. Specifically, the stylesheet in this example invokes the user-defined `valueOf()` functions. For more information on using extensions with Intel XML Software Suite, see XSLT Extensions.
Redirect	Uses the Xalan-specific redirect extension [9]. The example shows that instead of writing the entire result of transformation into a single `Result` object, you can divide the output into portions and write them to the specified files.
MultiThread	Demonstrates transformation of input data in several threads.

For the Intel® XPath Accelerator

ApplyXPath	Evaluates an XPath expression over an input XML source.
XPathVariable	Evaluates an XPath expression with variables over an XML source.
XPathExternalFunction	Evaluates an XPath expression referring to an external function over a pre-built DOM tree.

Version Query Samples

VersionQuery	Gets the Intel XML Software Suite version information as structure or string description.

Multi-component Samples

BookStore	Demonstrates how to use Intel DOM parsing, validation, XPath, and XSLT functionality in an sample workflow of online book purchasing.
SoapMessage	Shows how to use Intel XPath and validation capabilities on a SOAP message.

3 Using the Intel® XML Parsing Accelerator

The Intel® XML Parsing Accelerator for C/C++ environment parses XML data as sequences of events (the SAX model), as DOM nodes or as internal data streams, and can evaluate input data against a set of constraints by using the Intel® XML Schema Accelerator.

Parsing Data in SAX Mode

To parse a document in the SAX mode, you need to define the set of actions to be executed when a certain part of an XML document is parsed. The Intel® XML Parsing Accelerator supports an interface that provides events similar to those defined in the ContentHandle interface of the SAX 2.0 standard [5].

To implement the event handle, create a new event handler class inheriting from the `ContentHandler` class. The `ContentHandler` class provides major required virtual methods for parsing events. Your code can inherit from the class and implement the methods required for your application. In Example 1, a new handler is created.

Example 1. Creating a New Handler for Data Processing

```
class SAX2CoreDumper : public ContentHandler {
public:
virtual void startNamespace(unsigned int newNamespaceTag,
......
}
```

Follow these steps to parse your input data in SAX mode:

1. Get a SAX parser factory object.

   The Intel XML Parsing Accelerator maintains a SAX implementation factory for all `SAXParser` objects. Call the `newInstance()` method to create a SAX parser factory.

2. Create the parser.

   Call the method `createSAXParser()` in the `SAXParserFactory` class to create a new `SAXParser` object.

3. Register an event handler object.

The SAXParser class provides the method `setContentHandler()` to register the user-defined event handler object.

NOTE

A missing event handler raises an exception.

4. Perform the SAX parsing process.

   Call the `parse()` method of the SAXParser instance to parse an XML source and execute all event actions on the inherited `ContentHandler` object. Alternatively, you can call another `parse()` method providing a `FileReader`, `MemoryReader`, or another object inheriting from the `Reader` object.

5. Release the objects involved in the parsing process.

   Call the following methods of the SAXParserFactory class:

   a. `releaseSAXParser()` to release a SAXParser object.

   b. `releaseInstance()` to release a SAXParserFactory object.

Example 2 illustrates the process of parsing a document in the SAX mode.

Example 2. Parsing an XML File in SAX Mode

```
//Get a SAX parser factory.
SAXParserFactory* saximpl = SAXParserFactory::newInstance();

//Create SAXParser from the SAX parser factory.
SAXParser* parser = saximpl->createSAXParser();

//Create SAX2CoreDumper to dump XML events to XML content.
SAX2CoreDumper sax2CoreDumper(fout);

//Set SAX2CoreDumper as ContentHandler into SAXparser.
parser->setContentHandler(&sax2CoreDumper);

//Parse XML file through the file URI.
parser->parse(xmlFileName);

//Release the SAXParser created by SAXFactory.
saximpl->releaseSAXParser(parser);
```

```
//Release the SAXParserFactory.
SAXParserFactory::releaseInstance(saximpl);
```

Parsing Data in DOM Mode

Follow these steps to parse your input data in DOM mode:

1. Get a DOM implementation factory object.

   The system manages the DOM implementation factory for all `DOMImplementation` objects. Call the `newInstance()` method to create a DOM parser factory.

2. Get a DOM implementation object.

   To conform to the DOM 2.0 Core standard [4], the Intel® XML Parsing Accelerator provides another `DOMImplementation` class , which is managed by `DOMImplementationFactory`. Call the `getDOMImplementation()` method to create a DOM implementation object.

3. Create the parser.

   Call the method `createDOMParser()` in the `DOMImplementation` class to get a new `DOMParser` object.

4. Perform the DOM parsing process.

   Call the method `parse()` of the `DOMParser` object to parse the XML source and create Document tree object. Alternatively, you can call another `parse()` method providing a `FileReader`, `MemoryReader`, or another object inherited from the `Reader` object.

5. Release the objects involved in the parsing process.

   Call the following methods of the `DOMImplementation` class:

   a. `releaseDOMParser()` to release a `DOMParser` object.

   b. `releaseDocument()` to release a `Document` object.

Example 3 illustrates the process of parsing XML data in the DOM mode.

Example 3. Parsing an XML file to a DOM Tree

```
// Get the DOM implementation factory.
DOMImplementationFactory* domfactory =
DOMImplementationFactory::newInstance();
```

```
// Get DOMImplementation managed by DOMImplementationFactory;
// do not release it.
DOMImplementation* domimpl = domfactory->getDOMImplementation();

// Create a DOMParser form DOM Implementation.
DOMParser* parser = domimpl->createDOMParser();

// Parse the XML file to a DOM tree through the file URI.
doc = parser->parse(xmlFileName);
if(doc!=NULL)
domimpl->releaseDocument(doc);

// Release the DOMParser created by DOMFactory.
domimpl->releaseDOMParser(parser);

// Release DOMImplementationFactory.
DOMImplementationFactory::releaseInstance(domfactory);
```

You can further access and update the DOM tree created by `DOMParser`; in Example 4, a new element with a local name `employeeId` is created belonging to the `test` namespace, and then appending the element to a document object.

Example 4. Editing a Document Object

```
// Access the child node that is the root element of the document
doc.
Element* docElem = doc->getDocumentElement();

// Create an element object.
Element* element=doc->createElementNS("test","employeeId");

// Append the element object as the last child of the root
element.
Node* appendedChild = docElem->appendChild(element);
```

Parsing Data in XMLDocument Mode

In the `XMLDocument` mode, the Intel® XML Parsing Accelerator uses `XMLDocument`, an Intel proprietary compact description format of an XML document. This format is an efficient intermediate XML data representation across different components of the Intel XML Software Suite.

Follow these steps to parse your input data in the `XMLDocument` mode:

1. Get an `XMLDocument` parser factory object.

   The system manages the `XMLDocument` parser factory for all `XMLDocParser` objects. Call the `getFactory()` method to get the handle to the factory.

2. Create the parser.

   The method `createXMLDocParser()` in the `XMLDocFactory` class provides the new `XMLDocParser` object that you can use.

3. Perform the parsing.

   Call the method `parse()` of the `XMLDocParser` to parse the XML source and creates an XMLDocument object. Alternatively, you can call another parse() method providing a `FileReader`, `MemoryReader`, or another object inheriting from the `Reader` object.

4. Release the objects involved in the parsing process.

   Call the following methods of the `XMLDocFactory` class:

   a. `releaseXMLDocParser()` to release an `XMLDocParser` object.

   b. `releaseXMLDocument()` to release an `XMLDocument` object.

   c. `releaseInstance()` to release the `XMLDocFactory` object.

Example 5 illustrates the process of parsing XML data in the `XMLDocument` mode.

Example 5. Parsing an XML Document in the XMLDocument mode

```
// Get the XMLDocument implementation factory.
XMLDocFactory* docFactory = XMLDocFactory::newInstance();

//Create XMLDocParser from XMLDocFactory.
XMLDocParser* parser = docFactory->createXMLDocParser();

// Parse the XML file to an XmlDcoument object through the file
URI.
doc = parser->parse(xmlFileName);

//Release the DOM tree created by DOMParser.
if(doc!=NULL)
```

```
docFactory->releaseXMLDocument(doc);

//Release the DOMParser created by DOMFactory.
docFactory->releaseXMLDocParser(parser);

//Release XMLDocFactory.
XMLDocFactory::releaseInstance(docFactory);
```

Parsing Data with Schema Validation

The Intel® XML Parsing Accelerator can operate in the non-validating and validating modes. Follow these steps to enable the Intel XML Parsing Accelerator with schema validation provided by the Intel® XML Schema Accelerator:

1. Implement a `Schema` object, as described in steps 1 and 2 of the Validating Data section.

2. Enable schema validation by setting the `Schema` object into the parser via the `setSchema()` method, for example:

   ```
   pSaxPar->setSchema(ps);
   ```

   Initially, the `Schema` object is set to `NULL` and `setSchema()` changes it. When a schema contains a non-null value, the `Parser` object uses the `Validator` object created from it to validate documents before passing information to the application. When validation finds errors, the `Parser` object handles them as if those errors were found by the `Parser` object itself. In other words, if the user-specified `ErrorHandler` is set through `setErrorHandler()` in `SchemaFactory`, the errors will go to it.

   To cancel validation in the next round of data parsing, call `setSchema(0)`.

3. Parse your data with validation enabled.

   Call the method `parse()` of `Parser` to process the document. For example:

   ```
   pSaxPar->parse(xmlfile);
   ```

For details on the Intel XML Schema Accelerator specifics, please see the Using the Intel® XML Schema Accelerator section.

Configuring the Intel® XML Parsing Accelerator

User applications can register a custom entity resolver in SAX, DOM or XMLDocument parser to resolve system identifiers and open connections to entities.

Entity Resolver

This section uses the SAX parser as an example. DOM and `XMLDocument` parsing have almost the same steps.

By default, the Intel® XML Parsing Accelerator resolves entities that it encounters in XML data by the provided URL. Many SAX applications will not need to customize their entity resolver, but it will be especially useful for applications that build XML documents from databases or other specialized input sources, or for applications that use URI types other than URLs. If this is your case, implement the `EntityResolver` interface and override the `resolveEntity()` method.

```
class MyEntityResolver : public EntityResolver
  {
  …resolveEntity(…)
    {
    // your own implementation
    }
  }
```

When customizing is finished, register the entity resolver to a SAX parser before the parsing process.

```
MyEntityResolver myEntityResolver;
saxparser. setEntityResolver(&myEntityResolver);
```

Namespace Awareness

By default, the Intel® XML Parsing Accelerator processes markup vocabularies with namespace-aware enabled, which conforms to W3C* Namespaces in XML 1.0 Recommendation [2]. For backward compatibility, the Intel XML Parsing Accelerator provides switch and query interfaces to set and get namespace aware setting:

- `setNamespaceAware(bool namespaceAware)` configures the parser to enable or disable XML namespace aware. By default the value of the parameter is set true.
- `isNamespaceAware()` queries the status of the current setting on namespace awareness. You need to enable namespace awareness before starting the parsing process.

4 Using the Intel® XML Schema Accelerator

The Intel® XML Schema Accelerator for C/C++ is responsible for verifying input XML data against a *schema*, which is a set of constraints that can be checked or enforced against an XML document. The Intel XML Schema Accelerator can be invoked by the Intel® XML Parsing Accelerator when performing DOM and SAX parsing, as described in section "Using the Intel® XML Parsing Accelerator".

Validating Data

To run the Intel® XML Schema Accelerator for C++ against your data, do the following:

1. Get a new `SchemaFactory` object using the `newInstance(XMLChar* language)` method. Application code must release the object after using it.

   Method `newInstance()` is thread-safe: it returns a new `SchemaFactory` object for every call. However, the `SchemaFactory` object is not thread-safe because it provides `set` functions, so that different threads can modify the properties of the same `SchemaFactory` object.

   After obtaining the `SchemaFactory` object, you can set customized `ErrorHandler` or `ResourceResolver`, as described in the Configuring the Intel® XML Schema Accelerator section. If you do not set those options, the Intel XML Schema Accelerator compiles the schema XML document with the default behavior.

2. Load and compile the schema to be used.

   The process of compiling a schema starts when you create it. You can create a schema by calling one of the following methods of the `SchemaFactory` class:

   - `createSchema()` creates a special `Schema` object and during the validation process, looks for a set of constraints in the attributes `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation` of the processed XML document.

NOTE

Specifying the location of the schema using the attributes `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation` is not recommended because these schema location hints can introduce a vulnerability to denial-of-service attacks.

- `createSchema(XMLChar* systemId)` creates a `Schema` object according the `systemId` parameter specified.
- `createSchema(XMLChar* systemId[], unsigned size)` creates a `Schema` object according to the `systemId` array values specified. The Intel XML Schema Accelerator is responsible for creating a restriction set by combining the schemas pointed to by `systemId` array values.
- `createSchema(StreamSource* streamSource)` creates a `Schema` object according to the input stream source.

3. Get a validator from the schema.

   Create a `Validator` object via the `createValidator()` method of the `Schema` object. The `Validator` class provides the following methods for validating the XML document:

   - `validate(Source*)` validates the XML document. The method accepts pointers to `StreamSource`, `SAXSource`, `DOMSource`, `XMLDocSource`, and `Source` as the input parameters.
   - `validate(Source*, Result*)`supports only `SAXResult`. The method validates `SAXSource` and reports the events to the handlers registered in `SAXResult` during validation. If the `Result` object is `NULL`, this method acts as `validate(Source*)`. For other `Result` object input, it throws `IllegalArgumentException`.

NOTE

For the `validate(Source*, Result*)` method, the `Source` instance should match the `Result` instance.

`doValidation(Source*)` validates the XML document. The method accepts pointers to `StreamSource`, `SAXSource`, `DOMSource`, `XMLDocSource`, and `Source` as the input parameters. It has the same functionality as `validate(Source*)`. The difference is that it returns `false` instead of throwing exception in case of error. To view the error message, use the `getErrorMessage()` method.

`doValidation(Source*, Result*)` supports only `SAXResult`. This method validates `SAXSource` and reports the events to the handlers registered in

SAX`Result` during validation. If the Result object is `NULL`, this method acts as `doValidation(Source*)`. For other `Result` object input, it returns false. This method has the same functionality as `validate(Source*, Result*)`. The difference is that it returns `false` instead of throwing exception in case of an error. To view the error message, use the `getErrorMessage()` method.

It is recommended to use the `doValidation()` methods. They show better performance than the corresponding `validate()` methods in validating invalid XML instances, because the `doValidation()` methods do not have to go through the exception handling process which is less efficient than the simple "return `false`" mechanism.

`const XMLChar* getErrorMessage()` returns the error message content if the `doValidation()` method returns `false`. This method does not return the error message if a `validate()` method fails to validate the xml instance.

4. Parse the document you need to check.

5. Validate your data.

If the validate() method is used, and the source data is valid, the component displays nothing. Otherwise, the Intel XML Schema Accelerator throws `ValidationException` or reports errors to the specified custom `ErrorHandler`.

If the `doValidation()` method is used, the Intel XML Schema Accelerator returns TRUE or FALSE to show whether the source data is valid or not. To view error messages, use `getErrorMessage()` or the specified custom `ErrorHandler`.

The current version of the Intel XML Schema Accelerator does not support PSVI, so the output is not accompanied by augmented information.

Example 6 illustrates how to check data with the Intel XML Schema Accelerator.

Example 6. Validation Procedure

```
// Get system-managed schema implementation factory;
// do not release it.


SchemaFactory* schemaFactory =
SchemaFactory::newInstance("http://www.w3.org/2001/XMLSchema");
// Set the customized ErrorHandler and ResourceResolver, if
defined.
// schemaFactory->setErrorHandler(customizedErrorHandler);
// schemaFactory->setResourceResolver(customizedResouceResolver);
```

```
// Enable the full checking for schema file and this is time-
consuming.
schemaFactory-
>setFeature("http://xml.intel.com/validator/schema_full_checking"
, true);


// Create a Schema object from SchemaFactory.
Schema* schema = NULL;


try{
 schema = schemaFactory->createSchema("schemavalidator.xsd");
}
catch(ValidationException &e){
// handle the errors of schema compiling
}
Validator* validator = schema->createValidator();

// Specify the input, StreamSource in the example.
StreamSource streamSource("schemavalidator.xml");
bool result = validator->doValidation(&streamSource);
if(!result) {
// handle the error
printf("%s\n",validator->getErrorMessage());
}
// the validate method can also be used, as the following code
/*
 try{
validator->validate(&streamSource);
 }
 catch(ValidationException &e){
 // handle the errors of validation
 }
*/


// Release the Validator object created by Schema.
schema->releaseValidator(validator);


// Release the Schema object created by SchemaFactory.
```

```
SchemaFactory->releaseSchema(schema);
SchemaFactory::releaseInstance(schemaFactory);
```

Configuring the Intel® XML Schema Accelerator

User applications can employ several means of adjusting the validation process, as follows.

Error Handling

By default, the Intel® XML Schema Accelerator handles errors in the following way:

```
void warning(Exception & ex) {
//
}
void error(Exception & ex) {
}
void fatalError(SAXParseException ex) {
//
}
bool error(const XMLChar* err) {
return true;
}
```

When you set an error handler, errors found during parsing a schema or validating an XML document are first sent to the `ErrorHandler` object. During schema parsing, the function `error (Exception&)` is invoked if the `ErrorHandler` object is registered. The `error(Exception&)` and `bool error(const XMLChar* err)` error functions are invoked in validation run time.

The `bool error(const XMLChar* err)` function can be used to make the validation stop or continue when an error occurs. If FALSE is returned, the validation stops. If TRUE is returned, the validation continues until a fatal error, for example, a parsing error.

You can specify your own error handler instead of the default one. For that, you need to define the following functions:

- `void error(Exception)`
- `void fatalError(Exception)`
- `void warning(Exception)`
- `bool error(const XMLChar* err)`

Example 7 illustrates the procedure of setting a custom error handling behavior.

Example 7. Setting a Custom Error Handler

```
class CustomizedErrorHandler : public ErrorHandler {
void warning(Exception & ex) {
//
}
void error(Exception & ex) {
fprintf(stderr, "\nError:\n[%s]\n",
ex.getErrorMessage().c_str());
}
void fatalError(Exception &ex) {
//
}
bool error(const XMLChar* err) {
fprintf(stderr, “\nError:\n[%s]\n”, err);
return true;
}
}
```

NOTE

With this approach, the Intel XML Schema Accelerator does not distinguish warnings, errors and fatal errors, so that all errors are reported to `ErrorHandler::error()`.

External Resource Resolution

By default, the Intel XML Schema Accelerator resolves resources in the following way:

```
StreamSource* resolveResource(const XMLChar* nameSpace, const
XMLChar* resourceURI, const XMLChar* baseURI){
return NULL;
}
```

To redirect an included or imported file or the specified schema location, you can define your own external resource resolver by using the `ResourceResolver::resolve()` function, as shown in Example 8.

Example 8. Setting a Custom Resource Resolver

```
class CustomizedResourceResolver : public ResourceResolver {
StreamSource* resolveResource(const XMLChar* namespace, const
XMLChar* resourceURI, const XMLChar* baseURI) {
  //redirect the resource URI to other places.
  return NULL;
}
```

5 Using the Intel® XSLT Accelerator

The Intel® XSLT Accelerator for C/C++ performs transformation of input XML data in a variety of formats employing the XSL language and the additional capability of XSLT extensions belonging to the EXSLT set or defined by the user.

Performing the XSL Transformation

To transform your input data with the Intel® XSLT Accelerator, do the following:

1. Instantiate a `TransformFactory` object.

   To abstract your application from the internals of the transformer, use the abstract `TransformFactory` class with a static `newInstance()`method. This method returns a handle to a specific object of the underlying `TransformFactory` implementation.

2. Compile the stylesheet.

   Use the `newTemplates(Source* xslSource)`method in the `TransformFactory` class to compile a stylesheet. This method reads the supplied `Source` stylesheet, compiles the stylesheet, and produces a new `Templates` object.
   Using the `newTemplates()`method with no `Source` input produces a `Templates` object that creates transformers for identity transformation. Identity transformation makes an exact copy of input to output, but the data format can vary.

   Once compiled, the stylesheet is maintained as an internal representation of instructions and templates inside the `Templates` object. The compilation of the stylesheet is done only once, when the template is created.

   You can supply the stylesheet in a stream of XML markup `StreamSource`, in a DOM node `DOMSource`, or SAX input `SAXSource`, see `CompileStylesheet` in Sample Applications.

   You can register your own extension functions and the function resolver, and use your extension functions in the stylesheet, see XSLT Extensions. This function registration scheme is similar to that of the Intel® XPath Accelerator. Your function will be visible in all the `Transformer` objects created from this `Templates` object. For more details of extension function usage, see XSLT Extensions.

3. Set up the transformer.

a. Instantiate a `Transformer` object by using the `newTransformer()` method in the `Templates` class. This method returns an underlying implementation of the `Transformer` object.

b. If needed, configure the environmental settings for the transformer, see Configuring the Intel® XSLT Accelerator.

4. Perform the transformation.

   The `transform()` method of the `Transformer` object works over the XML source and places output in a `Result` object. Specifically, for each node in the XML source, the method does the following:

   1. Reads instructions in the `Templates` objects.
   2. Selects one of the following template options:
      - A template rule in the `Templates` object
      - The default template rule as specified in the XSLT specification
      - None

      The XML source can go in the form of a `StreamSource`, `DOMSource`, `SAXSource`, or `XMLDocSource` object, and the output can be a `StreamResult`, `DOMResult`, `SAXResult`, or `XMLDocResult` object. `StreamSource`, `DOMSource`, `SAXSource`, and `XMLDocSource` classes are inherited from the abstract `Source class`, and `StreamResult`, `DOMResult`, `SAXResult`, and `XMLDocResult` classes are inherited from the abstract `Result` class. For details, see Using Source and Result Types.

XSL transformation is illustrated by the `SimpleTransform` sample application that is supplied with the product, see Sample Applications. The step-by-step procedure for using the `Transformer` object to get a document object tree is illustrated in Example 9.

Example 9. Using a Transformer to get a DOM Tree

```
// Generate a TransformFactory object.
TransformFactory* tFactory = TransformFactory::newInstance();

//Compile the stylesheet, obtain the Templates object.
StreamSource ss ("foo.xsl");
Templates* templates= tFactory->newTemplates(&ss);

//Create the transformer.
Transformer* transformer = templates->newTransformer();
```

```
//Create Source and Result objects.
StreamSource inDoc("foo.xml");
DOMResult domResult();

//Perform the transformation.
transformer->transform(&inDoc, &domResult);

//Now you can get the output Node from DOMResult.
Node* node = domResult.getNode();

//Release the transformer, templates and factory.
templates->releaseTransformer(transformer);
tfactory->releaseTemplates(templates);
TransformFactory::releaseInstance(tfactory);
```

You can also use an extension function to redirect your output into one or more files, see the Redirect Extension section.

Customizing the Intel® XSLT Accelerator

Use the following options to customize the operation of the Intel® XSLT Accelerator:

- Set an external URI resolver by using `setURIResolver()` and `getURIResolver()` functions, see the *API Reference* in `doc/apidoc` folder for details.
- Set output properties in the `xsl:output` element of a stylesheet.

  After a stylesheet is compiled, the output properties that are specified in the XSLT stylesheet can be queried using the function `getOutputProperties()` on the `Templates` or the `Transformer` object, or `getOutputProperty()` on the `Transformer` object. You can override output property values when performing a transformation by calling functions `setOutputProperty()` and `setOutputProperties()` on the `Transformer` object.
- Add parameters to your stylesheet during transformation.

  For that, the `setParameter()` method of the `Transformer` object is used. After setting a parameter, you can retrieve it using the `getParameter()` function, see `StylesheetParameters` in Sample Applications.

- Enable parallel transformation of an XML document by multiple threads.

  For that, change the `XSLTParallelism` setting in the config.xml file as described in Configuring the Intel® XML Software Suite.

- Set the mode property to specify the target behavior that Intel XSLT Accelerator conforms to. You can choose between 5 modes: Standard, MSXML, XALANJ, XSLTC, and XT. The default value is Standard. You can set this property as follows:

```
<component name ="Intel XSLT Accelerator">
      <property name="mode" value="XALANJ" />
</component>
```

In this example, the Intel XSLT Accelerator conforms to the Xalan-J processor behavior.

Object Types

Supported Object Types

As shown in the transformation description, XML processing can involve data streams, DOM trees, SAX events and the Intel proprietary format `XMLDocument`. Each object type has its specifics in the transformation, as described below.

Streams

To create a `StreamSource` or a `StreamResult` object, you can use a system ID, which is a filename in the URI syntax, or an object: `Reader` for the source, `Writer` for the result.

DOM Trees

Your transformations can involve `DOMSource` and `DOMResult` objects. These involve operations with a DOM (document object model) tree.

To transform your data into a DOM tree, create a new `DOMResult` object or use `DOMResult.setNode()` to assign a new container.

SAX Events

You can use SAX (Simple API for XML) events in your input data, source stylesheet instructions or output. In the transformation engine, the `SAXParser` interface defines the `parse()` method to produce SAX events. When `parse()` is called, the parser

invokes one of the callback handler methods in your application or in the transformer. The SAX events are then handled by the callback handler.

XMLDocument

You can use an `XMLDocument` object as input or output, but not as a stylesheet source. You can get an `XMLDocument` object from `XMLDocResult` and set it as `XMLDocSource` for another transformer. Usually, transforming input `XMLDocument` is faster than other source types. Therefore, using `XMLDocument` as an intermediate result in multiple or pipelined transformations may accelerate the process. For an example of such a transformation, see the example `TransformChain`.

XSLT Extensions

You can expand the functionality of XSLT transformations through the use of XSLT extensions. An extension can be a function or an element. The Intel® XSLT Accelerator component provides a subset extensions, functions or elements, as defined by the EXSLT community project [12]. Further in the document, these functions are referred to as *EXSLT extensions*.

Extensions can also be user-defined, that is, provided by the end-user code. These are limited to extension functions. The Intel XSLT Accelerator supports user-defined extension functions implemented in the C++ programming language. Specifically, an extension function is an `evaluate()` function of a C++ class that is invoked during the execution of an XSLT transformation.

This section provides an overview of supported extensions and their usage.

EXSLT Extensions

The current version of the Intel® XSLT Accelerator supports EXSLT extensions grouped into the following modules:

- *Common* functions cover the basic operations; in the current version, the Intel XSLT Accelerator supports only one function of this group, `common:node-set`, which returns a node set from a result tree fragment. The Apache* Xalan*-specific `nodeset` extension matching the EXSLT `common:node-set` function is also supported.

- *Date-and-time* functions handle operations related to date and time; for example, the `date:hour-in-day` function returns the hour of the day as a number.

  The current version of the Intel XSLT Accelerator supports all core functions and most other functions as defined in the EXSLT resource [12], specifically:

```
date:add
date:add-duration
date:day-abbreviation
date:day-of-week-in-month
date:day-in-month
date:day-in-week
date:day-in-year
date:day-name
date:date
date:date-time
date:difference
date:duration
date:hour-in-day
date:leap-year
date:minute-in-hour
date:month-abbreviation
date:month-in-year
date:month-name
date:second-in-minute
date:seconds
date:sum
date:time
date:week-in-month
date:week-in-year
date:year
```

The current version of the product does not implement the following EXSLT functions of this module: `date:parse-date`, `date:format-date`.

- *Math* functions provide facilities for performing mathematical operations; for example, the `math:max` function returns the maximum value of the nodes passed as the argument.

  The current version of the Intel XSLT Accelerator supports all functions of this module as defined in the EXSLT resource [12], specifically:

```
math:abs
math:acos
math:asin
math:atan
```

```
math:atan2
math:constant
math:cos
math:exp
math:highest
math:log
math:lowest
math:max
math:min
math:power
math:random
math:sin
math:sqrt
math:tan
```

- *Sets* functions allow you to manipulate node sets; for example, the `set:difference` function gets nodes of two sets as arguments, compares them and returns the difference between the two sets.

  The current version of the Intel XSLT Accelerator supports all functions of this module as defined in the EXSLT resource [12], specifically:

```
set:difference
set:distinct
set:has-same-node
set:intersection
set:leading
set:trailing
```

- *Strings* functions are responsible for string manipulation; for example, the `str:tokenize` function splits up a string and returns a node set of token elements, each containing one token from the string.

  The current version of the Intel XSLT Accelerator supports most functions of this module as defined in the EXSLT resource [12]; specifically:

```
str:align
str:concat
str:padding
str:split
str:tokenize
```

The current version of the product does not implement the following EXSLT functions of this module: `str:replace`, `str:encode-uri`, and `str:decode-uri`.

The current version of the Intel XSLT Accelerator does not support the following EXSLT modules: `dynamic`, `functions`, `random`, and regular expressions.

NOTE

The Intel XSLT Accelerator does not support alternative language implementations of EXSLT functions.

For detailed instructions on how to use EXSLT functions, see the EXSLT website [12]. This section tells you how to call an EXSLT function. Follow the steps below:

1. Declare a namespace referring to the EXSLT module that contains the desired function. For example, to call a date-and-time function, type:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:date="http://exslt.org/dates-and-times">
...
</xsl:stylesheet>
```

2. Call the extension function in your templates. For example, get the year as a number type from the input string, call the `date:year` function in the following way:

```
<xsl:value-of select="date:year()">
```

NOTE

By default, the extension namespace is output into the result tree. To prevent this, specify the `extension-element-prefixes` attribute value with the module namespace prefix.

User-Defined Extension Functions

User-defined extension functions enable you to augment transformations with custom functions. Extension functions can operate with several XSLT and non-XSLT object types as input arguments and return values. For an example of a user-defined extension function, see UDF in Sample Applications.

You must organize your extension functions with a function resolver. XSLT will consult the function resolver when it encounters an external function. The function resolver identifies and returns the handle to the corresponding function implementation or

NULL if it cannot find one. During evaluation of the expression that contains the function, the `evaluate()` method of the returned function implementation will be called.

To use a custom extension function, do the following:

1. Define your function.

   Create your functions classes by inheriting the abstract class `XFunction` and implementing the `evaluate()`method. The `evaluate()` method takes as arguments a list of input `XObject` object and an output `XObject`. `XObject` can contain an XSL data type (Boolean, string, number, node-set) or a non-XSL type. For details on how to get the value of an `XObject`, please see the `API Reference` in the `doc/apidoc` folder.

2. Define the function resolver.

   Create the resolver class inheriting the abstract `XFunctionResolver` class and implement its `resolveFunction()`method. This method takes the function URI, name and arity as input, and returns the needed `XFunction` implementation you previously defined. To implement this method, you can employ an algorithm, such as looking up the function signature in the hash table.

   In the current release, no default function resolver is provided but you can use and modify the sample implementation in the UDF example in sample applications. See a detailed definition of `XFunctionResolver` in the *API Reference*.

   Example 10 illustrates function resolution, with one external function `valueOf()` that accepts three arguments with URI `http://www.abc.com/funcs`. The example shows how to define this function and the function resolver.

3. Register your function resolver.

   To register your function resolver, use `registerExternalFunctionResolver(XFunctionResolver* extFuncResolver)` method in the `Templates` class. For Example 10, you can register your function in the following way:

   ```
   MyFunctionResolver funcResolver;

   templates-
   >registerExternalFunctionResolver(&funcResolver);
   ```

4. Declare the namespace for the function(s), for example:

   ```
   <xsl:stylesheet version="1.0"

   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   ```

```
xmlns:my-function="http://www.abc.com/funcs">
```

The Intel XSLT Accelerator supports Apache* Xalan*-specific namespace for the `node-set` function. You can declare the namespace `xmlns:xalan="http://xml.apache.org/xalan"`, and use the function `xalan:node-set` in your stylesheet, see the Xalan website [9] for details.

5. Call the extension function in your templates, for example:

```
<xsl:variable name="new-pop"

select="my-function:valueOf(`Hello',./name,'Welcome to
Intel!')"/>
```

NOTE

You can call an extension function inside another extension function both for user-defined and for EXSLT extension functions.

Example 10. External Function Resolution

```
class MyValueOf : public XFunction {
public:
bool evaluate(const iutil::XVector<XObject*>&
argumentList,XObject* result){
//Your implementation here
return true;
}
};
class MyFunctionResolver : public XFunctionResolver {
public:
XFunction* resolveFunction(const XMLChar* funcUri, const XMLChar*
functionName,int arity) {
if (strcmp(funcUri,"http://www.abc.com/funcs")==0
&& strcmp(functionName,"valueOf")==0
&& arity == 3)
return &_funcValueOf;
return NULL;
}
private:
MyValueOf _funcValueOf;
};
```

Other Extensions

This part groups any other extensions supported by the Intel® XSLT Accelerator that are not EXSLT or user-defined extensions.

Redirect Extension

This is an extension specific for Apache Xalan-Java* implementation [9] that enables you to redirect parts of the transformation output to one or more files. You can see how this extension is used by running the `Redirect` application in the `Examples` directory. For instructions on running this example, see Sample Applications.

6 Using the Intel® XPath Accelerator

The Intel® XPath Accelerator for C/C++ performs evaluation of XPath expressions against input data. An *XPath* expression is similar to a path as in URL, and can include expressions to manipulate strings, numbers and Boolean values; for example, the expression `//book/title[author="Charles Dickens"]` finds the titles of books written by Charles Dickens in a given XML document.

Performing the XPath Evaluation

This section illustrates how to perform the XPath Evaluation with the Intel® XPath Accelerator.

For details on setting up the Intel XPath Accelerator, see Configuring the Intel® XPath Accelerator.

To evaluate your input data with the Intel XPath Accelerator, do the following:

1. Instantiate an `XPathFactory` object.

   Call static method `newInstance()` of the `XPathFactory` class to get a pointer to the specific object of the underlying `XPathFactory` implementation. Calling this method, specify the object model. Currently two object models are supported:

   - `XPATH_DEFAULT_OBJECT_MODEL` uses DOM as internal representation with a limitation on the input file size.
   - `XPATH_INTEL_XML_DOCUMENT_MODEL` accepts large files over 1GB.

   An `XPathFactory` object and its derived `XPath` and `XPathExpression` objects can support only one object model once it is instantiated. Calling this method without an argument is equivalent to calling `newInstance()` of the `XPATH_DEFAULT_OBJECT_MODEL`.

2. [optional] Adjust the evaluation settings of the Intel XPath Accelerator.

3. Instantiate an `XPath` object.

   Call the `newXPath()` method of the `XPathFactory` object to create an instance of the underlying `XPath` evaluator implementation. An `XPath` object provides `evaluate()` method series to evaluate given XPath expressions.

   At this stage, you can set the namespace context, variable resolver, and function resolver for this `XPath` object.

4. [optional] Compile the XPath expression.

   Call the `compile()`method of the `XPath` object to compile an XPath expression. This method reads the supplied expression, compiles it and produces a new `XPathExpression` object. Once compiled, the expression is maintained as an internal representation of instructions in an `XPathExpression` object.

5. Evaluate the `XPath` expression.

   You can invoke the `evaluate()` methods in the `XPathExpression` class to evaluate a pre-compiled XPath expression or the `evaluate()` methods in the `XPath` class to evaluate a non-compiled XPath expression directly. These methods take a DOM node or a derived instance of `Source (StreamSource`, `DOMSource`, `SAXSource` or `XMLDocSource)` as input, though only some of them are accepted by a specific object model.

   The `evaluate()` methods return `XObject`, which describes the result of an XPath evaluation. The `XObject` object contains the result of the type specified by the XPath expression, but can implicitly convert it into a different type, according to the XPath 1.0 recommendation [8].

   For example, to convert the result node set into a string, call `getStringValue()`. To get the evaluation results as a string, you can also call the `evaluateAsString()` methods. Do not delete the `XObject` object or objects created from it.

6. Recycle all objects created by the `XPath` object and the used memory by calling the `purgeMemory()` method. After this method, all created `XPathExpression`, `XObject`, `XMLDocNodeIterator`, `DOMNodeIterator` and their related objects no longer exist.

7. Release the created `XPath` and `XPathFactory` object.

   Call the `releaseXPath()` method to release all `XPath` objects created from this `XPathFactory` object, and then call `releaseInstance()` to release `XPathFactory`.

The code in Example 11 shows how to evaluate a compiled XPath expression with a stream source and return a node set.

Example 11. Sample XPath Evaluation

```
//Create an XPathFactory object supporting the default object
model.
XPathFactory* xFactory = XPathFactory::newInstance ();


//Create an XPath object.
```

```
XPath *xPath = xFactory->newXPath();
//Compile the XPath expression.
XPathExpression* xPathExpression = xpath->compile ("/a/b/c");

//Evaluate it with a stream input source.
StreamSource inDoc("foo.xml");
XObject *xObject = xPathExpression->evaluate (&inDoc);

//Check the result type and value.
switch (xObject->getType()) {
case X_DOMNODESET:

//Iterate over the node set
for (DOMNodesetIterator *it=xObject->getDOMNodeIterator(); it &&
!it->end(); it->next()) {
idom::Node *node = it->element();
}
break;
case X_STRING:
case X_NUMBER:
case X_BOOLEAN:
break;
}

//Release all XPath objects.
xPath->purgeMemory();
xFactory->releaseXPath(xPath);
XPathFactory::releaseInstance(xFactory);
```

Example 12. Sample XPath Evaluation with `INTEL_XML_DOCUMENT_MODEL`

```
//First get a XMLDocSource
XMLDocFactory* docFactory = XMLDocFactory::newInstance();
XMLDocParser* parser = docFactory->createXMLDocParser();
XMLDocument* xmldoc = parser->parse(“foo.xml”);
XMLDocSource xmldocSource;
xmldocSource.setXMLDocument(xmldoc);
//Create an XPathFactory object supporting the
INTEL_XML_DOCUMENT_MODEL
```

```
XPathFactory* xFactory = XPathFactory::newInstance
(XPATH_INTEL_XML_DOCUMENT_MODEL);
//Create an XPath object.
XPath *xpath = xFactory->newXPath();
//Compile the XPath expression.
XPathExpression* xPathExpression = xpath->compile ("/a/b/c");
//Evaluate it with a XMLDocSource
XObject *xObject = xPathExpression->evaluate (&xmldocSource);
```

Configuring the Intel® XPath Accelerator

You can employ a number of means to adjust operation of the Intel® XPath Accelerator to your needs, as follows:

- Check whether a certain object model is supported by the specific `XPathFactory` object.

  Call `isObjectModelSupported()`: the object model can be `DEFAULT_OBJECT_MODEL` or `INTEL_XML_DOCUMENT_MODEL`. Table 5 defines available models and the types of input objects they support.

Table 5. Object Models of the Intel® XPath Accelerator

Model Name	DOMSource	SAXSource	StreamSource	XMLDocSource
`DEFAULT_OBJECT_MODEL` Uses DOM as the internal representation; file size limited due to memory consumption of DOM tree.	X	X Converted to `DOMSource`	X Converted to `DOMSource`	
`INTEL_XML_DOCUMENT_MODEL` Accepts large files up to and over 1GB.				X

- Get and set the supported granularity features.

  The Intel XPath Accelerator can support specific features, which you can handle by calling `getFeature()` and `setFeature()`. The string names of these features are predefined, and in the current version, only `FEATURE_SECURE_PROCESSING` is supported. This feature sets all references to the external functions in an XPath expression return an error. The default setting is FALSE.

- Set external resolvers for variables, namespace prefixes, and external functions.

To specify a resolver for an element referred to by the evaluated XPath expression, call `setVariableResolver()` and `setFunctionResolver()` before compiling and evaluating an XPath expression. The resolvers set with these methods become the default resolvers for any `XPath` objects created from this `XPathFactory` object. The variable resolver and function resolver can be set as properties of `XPathFactory` and `XPath` objects, and the namespace context can only be set as property of `XPath`. This section gives instructions on setting the resolvers and context.

Namespace Context

When namespace prefixes are used in an XPath expression, the prefixes need to be resolved to URI addresses. The mapping of a namespace prefix to a URI is provided by an object implementing the `NamespaceContext` interface. You can set your derived `NamespaceContext` objects to `XPath` objects by calling the `setNamespaceContext()` method. Example 13 demonstrates setting the namespace prefix mapping through the `getNamespaceURI()method`.

Example 13. User-defined Namespace Context

```
class MyNamespaceContext : public NamespaceContext {
public:
const XMLChar* getNamespaceURI(const XMLChar* prefix) {
if (strcmp (prefix, "user") == 0) {
return "http://user.com";
}
return NULL; //return empty string
}
};
```

Variable Resolver

When variables are used in an XPath expression, the Intel® XPath Accelerator needs to get the type and value of these variables. You can implement an `XPathVariableResolver` object and set the object via `setVariableResolver()` for these purposes. Objects `XPath` and `XPathFactory` have this method. If the `XPathVariableResolver` is set on `XPathFactory`, all `XPath` objects constructed from this `XPathFactory` will use the specified `XPathVariableResolver` by default.

The Intel XPath Accelerator uses the `XPathVariableResolver` to retrieve the value of a user-defined variable. A value of a variable must not change during an evaluation operation. The `XPathVariableResolver` interface provides method `resolveVariable()` returning the variable value according to the input name. For example, if you want to evaluate an XPath expression `//book[name=$var]`, you need to implement an `XPathVariableResolver` object, from which the `$var` can be retrieved. The code in the Intel XPath Accelerator shows how to use user-defined variables and the variable resolver. The code in Example 14 shows how to use user-defined variable resolver:

Example 14: Variable Resolver

```
class MyVariableResolver : public XPathVariableResolver {
public:
void resolveVariable(const XMLChar* varUri, const XMLChar*
varName, XObject* result) {
if (strcmp(varName, "var") == 0) {
result->setStringValue("Introduction to Algorithm");
}
}
};
```

Function Resolver

When user-defined functions are used in an XPath expression, the Intel® XPath Accelerator needs to access the value returned by the function. You can implement an `XFunctionResolver` object and set the object via `setFunctionResolver()` for these purposes. `XPath` and `XPathFactory` objects have this method. If the `XFunctionResolver` is set on the `XPathFacory`, all `XPath` objects constructed from this `XPathFactory` will use the specified `XFunctionResolver` by default.

The Intel XPath Accelerator uses the `XFunctionResolver` to retrieve the value of a user-defined function. The `XFunctionResolver` interface provides the `resolveFunction()method` returning the `XFunction` object according to the input function name and its arity. The `XFunction` object also requires you to provide the implementation, so that you could wrap the function code into an `XFunction` object. A sample `XPathFunctionResolver` for evaluating an XPath expression `"user:maximum(3,4)"` could be the one shown in Example 15.

Example 15: User-defined XPath Function Resolver

```
class MyFunction: public XFunction {
public:
bool evaluate(const iutil::XVector<XObject*>& argList,XObject*
result)
{
//The function implementation is here
double d1 = argList[0]->getNumberValue();
double d2 = argList[1]->getNumberValue();
result->setNumberValue(d1>d2?d1:d2);
}
};
class MyFunctionResolver: public XFunctionResolver {
static MyFunction myFunction;
public:
```

```
XFunction* resolveFunction(const XMLChar* funcUri, const XMLChar*
functionName,int arity) {
if (strcmp(funcUri, "http://user.com") == 0
&& strcmp(functionName,"maximum") == 0)
return &myFunction;
else
return NULL;
}
};
```

7 Troubleshooting

This section lists errors that you might encounter and suggested solutions.

Failed to install with the error message "This installer requires write privileges to the directory <license_dir>. Please ensure that you have these rights".

Check that you have access to the directory where the installation script is trying to place the license:

- On Linux* OS: `~/intel/licenses`
- On Windows* OS: `C:\Program Files\Common Files\Intel\Licenses`

Failed to launch the library.

Verify that your environment variables are configured to point to the correct locations. For details on setting the required variables, see Configuring the Intel® XML Software Suite. You can confirm the values for the path variables on the command line, for example:

- On Linux* OS:

```
echo $LD_LIBRARY_PATH
```

During compilation of an example, header files are not found.

- Check that the example code has not been moved from the original location.
- Make sure that the <**install_dir**>/include folder contains all header files.

On Windows* OS, failed to start application because `intel-xss.dll` is not found.

- Make sure that `intel-xss.dll` is in the same folder as the sample code.
- Check that you have installed the product as described in the *Installation Guide*.

On Linux* OS, during execution of the example, the following error is produced:
libintel-xss.so: Cannot open shared object file: No such file or directory

- Make sure that the `LD_LIBRARY_PATH` variable holds the value `<install_dir>/bin/ia32` or `<install_dir>/bin/intel64` depending on the platform.
- Make sure that `<install_dir>/bin` has all the files defined in `README`.

On Linux* OS in Eclipse* environment: compilation or linking fails.

Please check that the include path and the library path are correctly set, see Working With Eclipse IDE.

Appendix A: Acronyms and Definitions

This section lists the acronyms used throughout the document with their definitions.

Table 6. Acronyms Used in this Document

Acronym	Definition
API	Application Programming Interface
DOM	Document Object Model
EXSLT	Extensions to XSL Transformations
I/O	Input and Output
PSVI	Post-Schema-Validation Infoset
SAX	Simple API for XML
TrAX	Transformation API for XML Processing
XML	Extensible Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language

Appendix B: Compatibility Modes: Support for various Interpretations of XSL Conditions

This section provides details on the Intel XSLT Accelerator compatibility modes: Standard, XT, MSXML, Xalan, and XSLTC.

These modes allow you to simulate different XSL behavioral interpretation of third party XSLT processors, such as Saxon, MSXML, Xalan, XT, and XSLTC. You can switch the compatibility mode on and off by setting property parameters in the `config.xml` configuration file, located in `<installation directory>/conf directory`.

Table 6. Compatibility Modes: Support for various Interpretations of XSL Conditions

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
Local variable override	Two variables or parameters with the same name inside a template cause an error.	A local variable inside the scope of another local variable can have the same name. In this case, the inner variable shadows the outer variable; the outer variable retains its value.	Allowed in all modes: Standard, XT, Xalan, MSXML, and XSLTC.
Global variable override	A stylesheet with more than one binding of a global variable with the same name and import precedence causes an error.	You can declare a global variable more than once. The last declaration takes precedence.	Allowed in the Standard, XT, XalanJ and XSLTC modes. N/A in the MSXML mode.
Named template override	A stylesheet with more than one template with the same name and import	More than one named template can have the same name. The last declaration takes	Allowed in all modes.

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
	precedence causes an error.	precedence.	
Unknown variable	A variable not declared in the scope causes a static error.	You can use an undeclared variable if you skip static checking. An error only occurs when the variable's value is explicitly used at run time.	Allowed in the XT mode only. N/A in other modes.
Unknown named template	Unknown named template.	You can use an undeclared named template if you skip static checking. An error only occurs when the template is called during transformation.	Allowed in all modes.
Single right curly brace escape in AVT	Using a single right curly brace outside the expression in AVT causes an error. Use double left or right curly brace instead. They are automatically replaced by a single curly brace.	You can use a single right curly brace outside the expression. The result is the same as with two right curly braces.	Allowed in the XT mode only. N/A in other modes.
Complex content in comment PI and attribute construction	If xsl:processing-instruction and xsl:attribute create nodes other than text ones while instantiating the content of xsl:comment, an error occurs. The XSLT processor may signal	• If the offending node is a is a literal result element, the node and its content are ignored in the MSXML mode. In other modes	• Allowed with implementation defined behavior in the MSXML mode. • Partially allowed with

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
	an error or recover by ignoring the offending nodes and their content.	an error occurs. • If the offending node is constructed using xsl:element, the element is converted to a text node with the string-value of this element node. This is relevant for all modes. • If xsl:processing-instruction contains top-level XSLT instructions, an error occurs in the MSXML mode.	implementation defined behavior in Standard, XT, XalanJ and XSLTC modes. See the **Relax Rules** column for details.
Unrecognized top-level XSLT instruction	Only the XSLT elements defined in the specification and elements with non-null URI from other namespaces can be top-level elements.	• In the MSXML mode, an unrecognized top-level XSLT instruction causes an error. • In other modes, the unrecognized element is ignored.	Not allowed in the MSXML mode. Allowed in Standard, XT, XalanJ and XSLTC modes.
Unrecognized	If an element in a	In the modes other	Not allowed in

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
local XSLT instruction with forward compatibility	template is processed in forwards-compatible mode (with version>1.0), and XSLT 1.0 does not allow to initiate such elements, XSLT performs fallback.	than MSXML, an element not allowed in XSLT 1.0 causes an error if initiated at run time.	Standard, XT, XalanJ and XSLTC modes. N/A in the MSXML mode.
Invalid top-level element	Only the XSLT elements defined in the specification and elements with non-null URI from other namespaces can be top-level elements.	• In the MSXML mode, a top-level element outside the XSLT namespace with null URI.causes an error. • In other modes, the element is ignored.	Allowed in Standard, XT, XalanJ and XSLTC modes. Not allowed in the MSXML mode.
Unrecognized attribute on XSLT instruction	An XSLT instruction may have attributes not from other namespaces, if the expand-name of the attribute has a non-null namespace URI. Attributes with null namespace URI (for example, attributes with unprefixed names) not defined in the specification, cause an error for an XSLT instruction.	• If xsl:comment has any attribute in the MSXML mode (with both null and non-null URI), an error occurs. • If xsl:stylesheet or xsl:transform has an attribute 'xsl:version' instead of 'version' in	Not allowed with implementation-defined behavior in the MSXML mode. N/A in other modes.

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
		MSXML mode, an error occurs.	
Unrecognized XSLT attribute on literal result element	The literal result element has no attributes from the XSLT namespace.	The attributes in XSLT namespace not defined in the specification are ignored in the MSXML mode.	Allowed in all modes.
Invalid attribute value on XSLT instruction	The specification defines the value range or specifics of a certain attribute on an XSLT instruction.	• In the MSXML mode, an error occurs when: 1. xsl:output, method value is neither xml/html/text nor a valid QName 2. xsl:output 'standalone' is not defined 3. xsl:output 'indent' is not defined 4. xsl:template 'priority' is an empty string 5. xsl:sort 'data-type' is neither a number nor a text 6. xsl:sort 'order' is neither ascending nor descending	Allowed with implementation-defined behavior in Standard, XT, XalanJ, and XSLTC modes. Partially not allowed with implementation-defined behavior in the MSXML mode. See the **Relax Rules** column for details.

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
		7. xsl:sort 'case-order' is neither upper-first nor lower-first • In other modes, the attribute is not specified, and the errors described above are ignored.	
xsl:element 'name' is not a QName	If the string resulting from instantiating 'name' attribute is not a QName, an error occurs. An XSLT processor may signal the error or recover by ignoring the element itself and instantiate its content.	In the MSXML mode, the XSLT processor signals an error. In other modes, it recovers.	Allowed in Standard, XT, XalanJ and XSLTC modes. Not allowed in the MSXML mode.
Non-white space texts in xsl:attribute-set	XSLT instructions have restrictions on their contents.	In the MSXML mode, an error occurs when xsl:attribute-set contains non-white space texts.	Allowed in Standard, XT, XalanJ and XSLTC modes. Not allowed in the MSXML mode.
Invalid pattern	If the pattern specified in the template is invalid, an error occurs.	• In the MSXML mode, the pattern error is checked statically. If the pattern is invalid, it	Allowed in Standard, XT, XalanJ, and XSLTC modes. Not allowed in the MSXML mode.

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
		processor signals an error. • In other modes, the behavior is undefined.	
Output Namespace Alias stylesheet-prefix	Undefined in the specification.	The namespace specified as the stylesheet-prefix can be the output in all the modes, except for the MSXML mode.	Allowed in Standard, XT, XalanJ and XSLTC modes. Not allowed in the MSXML mode.
Nested comment	XML specification does not allow nested comments.	In the MSXML mode, comments such as <!--<!--abc--><!--> do not cause an error.	Allowed in the MSXML mode. N/A in other modes.
Undefined key	Undefined in the specification.	• In the MSXML mode, using an undefined key does not cause an error. • In other modes, it returns an empty nodeset.	Allowed in the Standard, XT, XalanJ and XSLTC modes. Not allowed in the MSXML mode.

Appendix C: References

This section lists reference to relevant external documents referenced in the current document.

1. W3C* XML 1.0 Recommendation, http://www.w3.org/TR/xml/
2. W3C Namespaces in XML 1.0 Recommendation, http://www.w3.org/TR/2006/REC-xml-names-20060816/
3. W3C XSLT 1.0 Recommendation, http://www.w3.org/TR/1999/REC-xslt-19991116.html
4. W3C Document Object Model (DOM) Level 2 Core Specification, http://www.w3.org/TR/DOM-Level-2-Core/; Level 3 http://www.w3.org/TR/DOM-Level-3-Core/
5. Simple API for XML (SAX), http://www.saxproject.org/
6. W3C XML Conformance Test Suite, http://www.w3.org/XML/Test/
7. W3C XML Schema 1.0, http://www.w3.org/TR/xmlschema-1/, http://www.w3.org/TR/xmlschema-2/
8. W3C XPath 1.0, http://www.w3.org/TR/1999/REC-xpath-19991116
9. Apache Xalan* Java* transformer, http://xml.apache.org/xalan-j/
10. Apache Xalan* C transformer, http://xml.apache.org/xalan-c/
11. OASIS* XSLT Conformance test suite, http://www.oasis-open.org/committees/documents.php?wg_abbrev=xslt
12. Extensions to XSLT, http://exslt.org/
13. Intel® XML Software Products website, www.intel.com/software/xml/
14. Eclipse IDE, http://www.eclipse.org/
15. Java* API for XML Processing (JAXP), https://jaxp.dev.java.net/

Index

D

DOM (document object model)..... 2

DOMParser..... 27

E

ErrorHandler 35

EXSLT extensions..... 42

Common 42

Date-and-time 42

Math 43

Sets 43

String 44

external resources

in schema validation 35

in XPath

function resolver..... 52

namespace context 51

variable resolver..... 52

I

Intel® XML Parsing Accelerator ...2

samples 23

using 26

in DOM mode 27

in SAX mode 26

in XMLDocument mode 29

Intel® XML Schema Accelerator ...2

samples..... 24

using..... 32

Intel® XML Software Suite..... 1

components of..... 1

features..... 4

quick start 9

Intel® XPath Accelerator 2

samples..... 24

using..... 48

Intel® XSLT Accelerator 2

configuring..... 40

samples..... 24

using..... 38

P

Parsing See Intel® XML Parsing Accelerator, using 2

S

Samples..... 10

SAX (simple API for XML)..... 2

event handle 26

SAXParser 26

Schema..... 32

SchemaFactory 32

Simple API for XML, See SAX..... 26

T

Templates 38

TransformFactory 38

V

validation 32

enabling 30

using, See Intel® XML Schema Accelerator 2

validator, See Intel® XML Schema Accelerator 2

X

XML Suite, See Intel® XML Software Suite 1

XMLDocument 2

XPath 48

XPathFactory 48

XSL (extensible stylesheet language) 38

extensions 42

EXSLT See EXST extensions 42

user-defined extensions See user-defined extensions 45

transformation 38